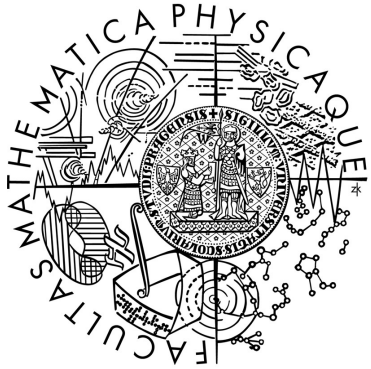
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Martin Major

Použití metod předpovídání budoucích uživatelských hodnocení pro doporučování filmů

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Martin Kruliš

Studijní program: Informatika
Studijní obor: programování

Praha 2012

Poděkování

Děkuji RNDr. Martinu Krulišovi za hodnotné rady a odborné vedení během mé práce. Dále bych chtěl poděkovat Martinu Pomothymu bez jehož svolení k využití dat ČSFD.cz by tato práce nemohla vzniknout.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 7. 12. 2012 Bc. Martin Major

Název práce: Použití metod předpovídání budoucích uživatelských hodnocení pro doporučování filmů

Autor: Bc. Martin Major

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Martin Kruliš

Abstrakt: Cílem práce je prozkoumat doporučovací algoritmy pro předpověď budoucích hodnocení filmů uživateli dle jejich předchozích hodnocení. Autor rozebere dostupné algoritmy a porovná úspěšnost vzorových implementací s vlastním algoritmem. Cílem je nalézt algoritmus pro co nejpřesnější předpovědi a zjistit, které parametry jsou pro předpověď důležité.

Klíčová slova: doporučovací systémy, kolaborativní filtrování, předpověď hodnocení, filmy, Netflix prize

Title: Application of User Ratings Prediction Methods for The Film Recommendations

Author: Bc. Martin Major

Department: Department of Software Engineering

Supervisor: RNDr. Martin Kruliš

Abstract: The aim of this work is to explore recommender systems for prediction user's future film ratings according to their previous ratings. Author will describe available algorithms and compare their results with his own algorithm. The goal is to find algorithm with the highest prediction accuracy and find the most important parameters for a good predictions.

Keywords: recommender systems, collaborative filtering, rating predication, films, Netflix prize

Obsah

1 Úvod

Od doby, kdy lidstvo začalo využívat znalosti jako svoji konkurenční výhodu, stoupá cena informací a dat. Po nástupu digitálního věku však začalo množství dostupných dat růst obřím tempem. Podle posledních výzkumů [1] se množství existujících dat více než zdvojnásobí každým rokem. Při takovém množství dat je již nejdůležitější schopností vytěžit z těchto dat maximální množství užitečných informací a ty využít pro získání nových vědomostí. V této práci rozeberu některé způsoby, pomocí kterých lze předpovídat uživatelská hodnocení, a porovnám jejich praktické implementace.

1.1 Základní pojmy

Obvyklý způsob vyhledávání ve velkém množství informací je prostřednictvím zadání určité vstupní informace. Například webový vyhledávač Google.com používá jako vstupní informaci krátký text zadaný uživatelem, zatímco třeba reverzní vyhledávač obrázků TinEye.com vyžaduje jako vstupní informaci obrázek. Když systém získá vstupní informaci, prohledá databázi předem indexovaných dat, porovná je se vstupem a na výstup vrátí indexovaná data, setříděná podle jejich podobnosti se vstupem. Tento systém lze samozřejmě dále rozšiřovat, lze brát v úvahu další doplňující informace, ale obecně lze konstatovat, že tento přístup je použitelný tam, kde je zapotřebí hledat v předem indexovaných datech taková, která jsou závislá na vstupní informaci.

Odlišný postup je nutné zvolit, pokud není dopředu známo, jaká data jsou hledána. Pokud je například nutné v neznámých datech najít souvislosti, je potřeba indexovaná data vhodným způsobem vyfiltrovat. Z toho vyplývá, že vstupní informací je zde zvolené filtrování a další vstupní informace již nejsou potřeba, případně slouží pouze pro upřesnění vyhledávání. Tento přístup se nazývá *informační filtrování*.

Související disciplínou jsou tzv. *doporučovací systémy*, které se snaží uživateli doporučit nějaký výrobek, službu, nebo něco jiného na základě předem

známých dat. Například systém, který uživateli doporučí film, který by se mu mohl líbit, nebo výrobek v e-shopu. Doporučovací systémy lze podle způsobu práce rozdělit do dvou tříd: obsahově zaměřené a uživatelsky zaměřené.

Obsahově zaměřené systémy fungují tak, že pro každý objekt v databázi (zboží, film,…) najdou několik podobných objektů. Když uživatel projeví zájem o některý z objektů (přidá si jej do košíku, kladně jej ohodnotí, apod.), systém mu nabídne podobný produkt. Podobnost produktů se určuje porovnáním jednotlivých parametrů, což někdy může být snadné, jindy to může být téměř nemožné (databáze úryvků básní).

Druhá třída doporučovacích systémů je zaměřena na uživatele a nazývá se *kolaborativní filtrování*. Tento přístup se snaží s pomocí informací o ostatních uživatelích v systému zjistit informace o aktuálním uživateli. Snahou je nalézt takové uživatele, jejichž parametry se co nejvíce podobají parametrům aktuálního uživatele. Poté je vysoká pravděpodobnost, že pokud se s některými uživateli shodoval v již známých datech, bude se s nimi shodovat i v ostatních. Aby tento přístup fungoval, musí být možné porovnávat jednotlivé uživatele a je potřeba dostatečné množství dat, aby bylo možné najít pro aktuálního uživatele dostatek podobných uživatelů. Nejčastějším využitím této techniky jsou e-shopy, které podle již prohlédnutého zboží hledají uživatele, kteří si prohlédli stejné zboží, a poté aktuálnímu uživateli nabídnou zboží, které si tito nalezení uživatelé již zakoupili.

1.2 Cíle práce

Cílem této práce je najít algoritmus, který uživatelům pomůže vybrat si film (např. v televizi, v kině,…), který se jim bude líbit. V případě úspěchu bude algoritmus použit na webu ČSFD.cz, který také poskytl testovací data. V této práci bychom chtěli dosáhnout dvou cílů. Za prvé předpovědět, jak se uživateli bude líbit určitý film a za druhé vybrat takový film, který by se mu mohl líbit. Je zřejmé, že druhou úlohu lze vyřešit iterací první úlohy nad množinou všech filmů a setříděním výsledků. Bohužel i s velmi rychlým algoritmem je provedení této úlohy výkonnostně velmi náročné. Proto se v praxi využije spíše doporučení

filmu z nějaké předem dané množiny, např. z filmů, právě uváděných v televizi, nebo v nejbližším kině.

Cílem tedy je nalézt co nejlepší algoritmus, který pro dvojici <uživatel, film> vrátí hodnocení, kterým by uživatel film ohodnotil.

Je nutné počítat s tím, že algoritmus nikdy nemůže být dokonalý, protože hodnocení filmu je velmi subjektivní záležitost a je ovlivněno mnoha dalšími faktory, např.:

- očekávání (pokud uživatel uvidí průměrný film s velikým očekáváním, pravděpodobně bude zklamaný a film ohodnotí špatně; pokud však uživatel čeká velmi špatný film a uvidí průměrný, pravděpodobně bude příjemně překvapen a film ohodnotí lépe než v prvním případu)
- forma (film zhlédnutý v kině je většinou hodnocen lépe, než stejný film zhlédnutý v televizi)
- nálada
- názor okolí (reklama a kamarádi mohou náš názor na film výrazně ovlivnit)

I přesto, že algoritmus určitě nebude dokonalý, určitě stojí za to, pokusit se vytvořit takový, který poskytne co nejlepší předpověď.

Je nutné zdůraznit, že cílem naší práce je předpovědět hodnocení filmu uživatelem, nikoliv obecné doporučení filmu uživateli. Pokud by cílem bylo obecnější „doporučování filmů“, pak určitě bude hrát velkou roli celkové zaměření uživatele. Specializovaný divák totiž sleduje ve svém oblíbeném žánru i méně kvalitní filmy. Průměrný divák naopak sleduje zpravidla spíše lepší díla, vybíraná kiny a televizními dramaturgy tak, aby se většině diváků líbila. Specializovanému divákovi by pravděpodobně udělalo radost, i doporučení filmu z jeho oblíbeného žánru, přestože nemá tak vysoké hodnocení. V této práci se však budeme zabývat pouze předpovídáním hodnocení filmu uživatelem, proto budeme takovéto případy uvažovat pouze okrajově.

1.3 Existující použití

Doporučovací systémy zažívají v poslední době rychlý rozvoj, protože dobře fungující doporučovací systém může firmě přinést výrazné zvýšení zisku.

Mezi první velké nasazení doporučovacího systému do reálného provozu patří **Amazon.com**, který uživatelům začal doporučovat výrobky na základě nákupu jiných uživatelů. Amazon.com totiž uchovává data o společném prodeji různých druhů výrobku. V databázi pak na základě získaných dat udržuje agregované informace o tom, které zboží bylo nejčastěji prodáváno současně s konkrétním výrobkem. Když poté prochází nový uživatel webem, Amazon.com si pamatuje navštívené stránky s výrobky. Poté najde výrobky, které se často prodávaly spolu s uživatelem prohlíženými výrobky, seřadí je dle četnosti výskytu a nabídne je uživateli jako doporučení.

Zde je popsán pouze základní princip. Doporučovací systém Amazonu je samozřejmě velmi komplexní, jeho detailní algoritmus je obchodním tajemstvím společnosti Amazon.com Inc. a je patentově chráněn [3].

Dalším příkladem doporučovacího systému je internetové rádio **Pandora.com**, které po zadání oblíbené hudby uživatelem dokáže vybírat hudbu podobnou. Doporučovací systém zde není založen na kolaborativním filtrování, ale na posuzování podobnosti jednotlivých písní. Základem tohoto projektu je systém Music Genome Project [4], kde tým odborníků ohodnotí pro každou píseň více než 450 atributů na škále 1 – 5 s krokem 0,5. Poté již systém dokáže porovnávat podobnost písní pouhým porovnáváním jednotlivých atributů.

Nejznámější systém pro doporučování filmů má pravděpodobně **Netflix.com**, což je největší on-line půjčovna filmů. Pro Netflix je velmi důležité, aby mohl po zhlédnutí filmu uživateli doporučit další film, který by se mu s co největší pravděpodobnostní mohl líbit. Zisk společnosti Netflix přímo závisí také na kvalitě jejich doporučovacího systému. Proto Netflix vyvinul vlastní algoritmus Cinematch, který slouží k doporučování filmů. Netflix navíc vypsal cenu Netflix prize (viz kapitola Netflix prize), kde nabízel cenu $1.000.000 týmu, který dokáže výsledky Cinematch zlepšit alespoň o 10%.

Existuje mnoho dalších firem, které využívají doporučovacích systémů, například:

- Last.FM – hudební databáze, přímo z hudebního přehrávače uživatele se odesílá informace o uživatelem poslouchané hudbě a podle toho mu Last.FM doporučuje další skladby.
- Synopsi.TV – doporučovací systém pro výběr filmů

1.3.1 Netflix Prize

Netflix Prize byla soutěž, vypsaná 2. října 2006 společností Netflix. Cílem bylo nalezení takového doporučovacího algoritmu, který by byl alespoň o 10% lepší, než jejich vlastní algoritmus Cinematch. Netflix uvolnil množinu testovacích dat obsahující tyto čtyři hodnoty: ID uživatele, ID filmu, datum hodnocení, hodnocení (na celočíselné škále 1-5). Množina testovacích dat obsahovala 100.480.507 hodnocení, které udělilo 480.189 uživatelů 17.770 filmům. Ke každému filmu byl dostupný název a rok uvedení. O uživatelích nebyly dostupné žádné údaje z důvodu ochrany soukromí. Z téhož důvodu byla některá hodnocení vynechána, některá pozměněna a některá uměle přidána. Dále byla uvolněna testovací množina s necelými 3 miliony záznamů, která obsahovala stejná data jako hlavní množina vyjma hodnocení uživatelů. Cílem týmů bylo co nejpřesněji předpovědět tato hodnocení na škále 1-5 (nemuselo být celočíselné). Celková kvalita předpovědi byla měřena jako střední kvadratická chyba (RMSE).

Jednoduchý algoritmus, který pouze spočítal průměrné hodnocení filmu a to předpovídal všem uživatelům dosáhl RMSE 1,0540. Algoritmus Cinematch, který používal Netflix dosáhl RMSE 0,9514, což je zlepšení o pouhých 9,73%. Cílem soutěže bylo zlepšení o 10% oproti algoritmu Cinematch, za což byla vyhlášena cena $1.000.000. Dokud se žádnému týmu nepodařilo dosáhnout cíle, byla každý rok vyhlašována průběžná cena $50.000 pro tým, který měl do té doby nejlepší výsledek, a zároveň došlo ke zlepšení alespoň o 1% od posledního vyhlášení průběžné ceny.

Ačkoliv byl algoritmus Cinematch poražen hned první týden po vyhlášení soutěže, zlepšování se zpomalilo a v letech 2007 a 2008 byly uděleny pouze průběžné ceny. Výsledná cena byla udělena až v roce 2009 týmu „BellKor's Pragmatic Chaos“, který vznikl spojením tří týmů.

Na příkladu Netflix prize je vidět, že oblast doporučování filmů je velmi obtížná. Přestože se soutěže zúčastnilo více než 2.000 týmů, podařilo se jim po třech letech dosáhnout celkového zlepšení o pouhých 18,84% oproti triviálnímu algoritmu. To ukazuje, že hodnocení filmu je velmi subjektivní záležitost, která je ovlivněna mnoha faktory, které není snadné předpovědět. Výsledný algoritmus je mixem více než 100 dílčích sub-algoritmů, což ukazuje, že neexistuje jeden snadný správný přístup.

Netflix po ukončení soutěže zveřejnil informaci [8], že výherní algoritmus v produkčním prostředí nepoužijí, neboť jej nelze dostatečně škálovat a kromě toho došlo v průběhu soutěže k postupné změně požadavků a vstupních informací.

1.4 ČSFD.cz

ČSFD.cz je internetový server zaměřený na filmy a TV seriály. Uživatelé si mohou na tomto serveru prohlížet informace o filmech a tvůrcích, mohou diskutovat v tematických diskuzích, prohlížet si program TV a kin a hlavně mohou hodnotit a komentovat jednotlivé filmy. Server má mnoho dalších funkcí, které však nejsou pro tuto práci důležité.

Na serveru ČSFD.cz existují tři základní entity, ke kterým jsou vázány ostatní informace a těmi jsou:

- film (souhrnné označení pro filmy, TV seriály, TV pořady, atd..)
- tvůrce (souhrnné označení pro režiséry, skladatele a herce)
- uživatel (registrovaný návštěvník ČSFD.cz)

1.4.1 Informace o filmu

O každém filmu mohou být na ČSFD.cz uvedeny tyto údaje:

- názvy filmu (originální název, přeložené názvy v jednotlivých zemích, festivalové názvy, neoficiální názvy)
- rok výroby filmu
- délka filmu v minutách
- země původu (i více)
- žánry filmu (i více)
- tvůrci filmu (režiséři, skladatelé, herci)
- související filmy (např. ostatní filmy ze stejné série)
- podobné filmy (filmy s podobným zaměřením)
- premiéry filmu (v kinech, na DVD a Blu-ray)
- uživatelské komentáře
- uživatelská hodnocení
- + mnoho dalších dat, která jsou již obtížně strojově zpracovatelná, či nejsou vhodná pro potřeby doporučování filmů

Ne o všech filmech jsou ovšem evidovány všechny tyto informace.

1.4.2 Hodnocení filmu

Každý uživatel, který se na ČSFD.cz zaregistruje, může hodnotit filmy na škále 1-5 hvězdiček + speciální hodnocení „*odpad!*“. V naší práci budeme hodnocení uvažovat na škále 0-100% vždy po kroku 20%, tedy „*odpad!*“= 0%, 1 hvězdička = 20% atd.

1.4.3 Srovnání s Netflix.com

Přesto, že se každý z těchto webů zabývá něčím jiným, obsahují podobná data. Oba weby si uchovávají informace o filmech, ale ČSFD.cz obsahuje informace o více filmech. V Netflix prize byla použita hodnocení k 17.770 filmům, na ČSFD.cz bylo ohodnoceno 131.349 filmů. Na druhou stranu bylo v Netflix prize uděleno přes 100.000.000 hodnocení a na ČSFD.cz pouze 40.000.000. Protože průměrný film má výrazně více hodnocení v Netflix prize,

může zde být predikce snazší a přesnější. V Netflix prize udělil průměrný uživatel 209 hodnocení, zatímco na ČSFD.cz to je 233 hodnocení.

Dalším rozdílem je hodnotící škála. Na Netflix.com je použita škála 1-5, na ČSFD.cz je používána škála 0-5. Proto nebude možné porovnávat RMSE absolutně, lze porovnávat pouze relativní zlepšení.

Dalším rozdílem mezi ČSFD.cz a Netflix prize je množství informací, které může algoritmus získat na vstup. Jednotlivé týmy v Netflix prize měly pouze velmi omezenou sadu dat, kterou mohly využít: jméno, rok vzniku filmu a hodnocení uživatelů (včetně data). Z názvu a roku vzniku filmu si mohly týmy (a mnoho to udělalo) stáhnout informace o filmu z jiného zdroje, např. IMDB.com. ČSFD.cz má oproti tomu sama k dispozici výrazně více informací. Není však jisté, zda tyto dodatečné informace mohou pomoci ke zlepšení hledaného algoritmu.

1.4.4 Spřízněné duše

Na ČSFD.cz nyní existuje funkce *Spřízněné duše*, která na základě podobného hodnocení filmů dokáže najít další uživatele s podobným vkusem. Při hledání spřízněných duší pro jednoho uživatele je potřeba porovnat jeho hodnocení filmů se všemi ostatními uživateli. Výkonnostní náročnost této funkce je velmi vysoká, protože pro každého uživatele je potřeba projít všechny ostatní uživatele, najít všechna společná hodnocení a porovnat je. Asymptotickou složitost výpočtu spřízněných duší pro jednoho uživatele můžeme tedy vyjádřit následovně:

$$O(u*r^2)$$ kde:

u je počet uživatelů,

r je počet hodnocení.

Pro tuto funkci je potřeba stanovit minimální množství ohodnocených filmů. Pokud bude minimální množství stanoveno jako příliš nízké, bude předpověď nepřesná kvůli malému vzorku dat. Pokud bude příliš vysoké, bude tuto funkci moci využívat jen malá část uživatelů. V současné době je limit stanoven na 500 ohodnocených filmů, což snižuje výkonnostní náročnost,

protože stačí prohledávat hodnocení pouze těch uživatelů, kteří splňují tuto podmínku.

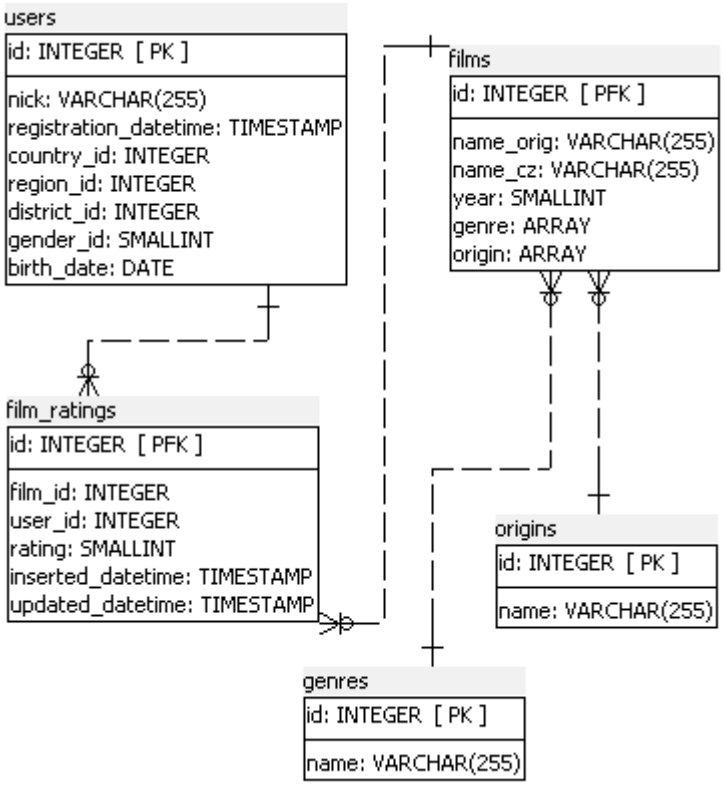
Složitost této funkce je bohužel příliš vysoká na to, aby ji bylo možné škálovat pro výrazně vyšší počet uživatelů a hodnocení. Navíc při každém dalším uděleném hodnocení není možné výsledky snadno upravit, ale musíme přepočítat vzájemnou podobnost mezi uživatelem, který udělil hodnocení, a všemi ostatními, což znemožňuje používat tyto výsledky v reálném čase pro doporučování filmů.

Z těchto důvodů je potřeba najít jinou funkci s nižšími výkonnostními nároky, která bude schopna nahradit přínosy této funkce. Ideální funkce by umožňovala nalézt uživatele, kteří mají podobný filmový vkus, a podle jejich hodnocení předpovídat, jak se bude vybranému uživateli líbit konkrétní film. Zároveň tato funkce musí umožnit dostatečné škálování.

2 Testovací data

Všechny testy byly prováděny na kompletních datech ČSFD.cz ke dni 9. 1. 2012. Množina testovacích dat zahrnovala 131.349 filmů a 39.780.898 hodnocení filmů, které udělilo 170.604 uživatelů (počítáme pouze s filmy, které někdo hodnotil a s uživateli, kteří udělili alespoň jedno hodnocení).

Data byla uložena v databázi PostgreSQL. Následující obrázek ukazuje strukturu tabulek v databázi:



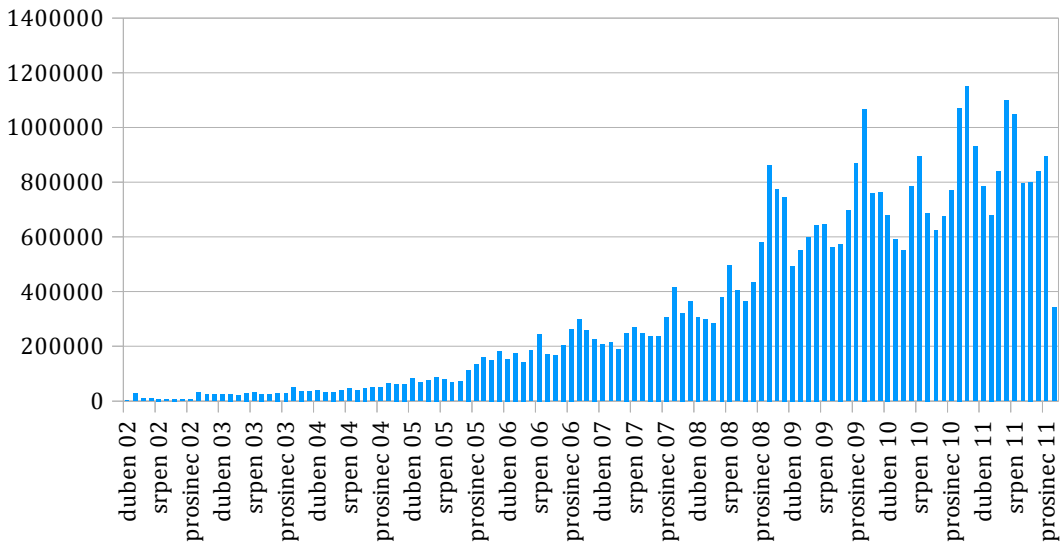
Obr. 1: Struktura tabulek v databázi.

Aby bylo možné posuzovat podobnost mezi uživateli, bude nutné počítat pouze s takovými uživateli, kteří již mají ohodnoceno určité množství filmů. Pokud některý uživatel ohodnotil pouze minimální množství filmů, není možné použít tato hodnocení ani k předpovědi jeho hodnocení, ani k předpovědi hodnocení ostatních uživatelů, protože při malém množství jím ohodnocených filmů nelze spolehlivě určit jeho filmový vkus uživatele. Jako limit zvolíme 500 ohodnocených filmů, což se může zdát na první pohled hodně, ale umožní nám

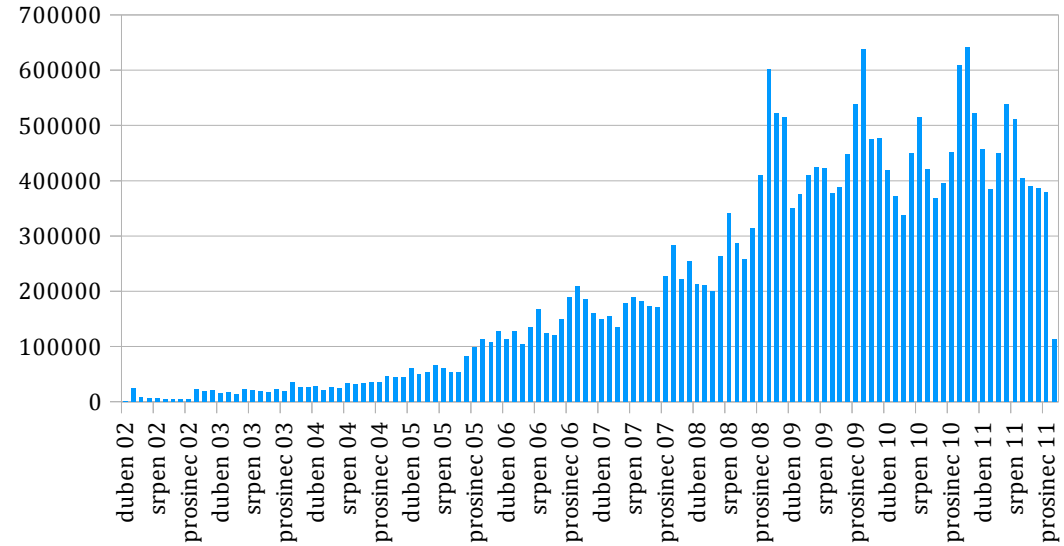
to lépe porovnávat jednotlivé uživatele a získat tak lepší výsledky. Kromě toho, 500 ohodnocených filmů je současný limit pro aktivování funkce *spřízněné duše*, popsané v minulé kapitole, takže uživatelé jsou již na tento limit zvyklí.

Aby bylo možné ověřovat úspěšnost jednotlivých metod, rozdělíme data do dvou skupin: *testovací data* a *kontrolní vzorek*. Testovací data obsahují 99% všech hodnocení a slouží k učení algoritmů. Kontrolní vzorek obsahuje 1% všech hodnocení a slouží k ověření úspěšnosti předpovědí. Přestože se kontrolní vzorek často vybírá náhodně, my zvolíme 1% nejnovějších dat. Taková množina dat lépe simuluje situaci, kdy na základě existujících hodnocení je potřeba předpovědět dosud neexistující hodnocení. Toto rozdělení odpovídá situaci, kdy u filmů, které dosud nebyly uvedeny do kin, existuje zatím málo hodnocení (pouze od novinářů z předpremiér). Hodně uživatelů je však zvědavých, jak se jim bude nový film líbit, a mají zájem o kvalifikované doporučení.

Po rozdělení na testovací a kontrolní vzorek je potřeba vybrat pouze ty uživatele, kteří mají v testovacích datech ohodnoceno alespoň 500 filmů. Ačkoliv by se mohlo zdát, že takto vysoká hranice diskvalifikuje velké množství uživatelů, nakonec tuto podmínku splnilo 24.792 uživatelů (budeme jim říkat *aktivní uživatelé*), kteří udělili dohromady 24.803.661 hodnocení. Ukázalo se tedy, že pouhých 15% uživatelů udělilo 62% všech hodnocení. To znamená, že i přes poměrně přísnou vstupní podmínku budeme počítat s většinou vstupních dat.

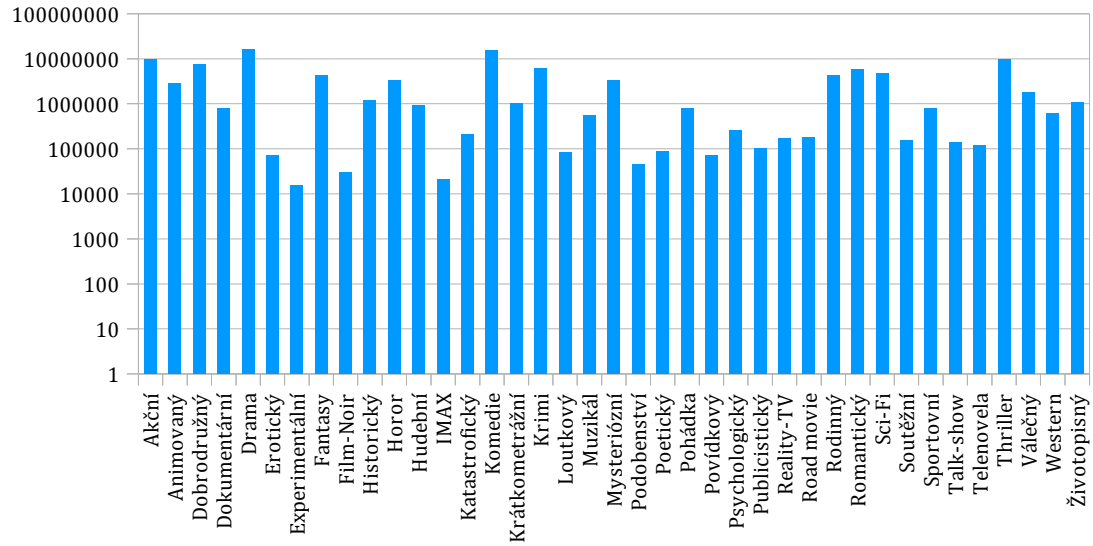


Obr. 2: Počet hodnocení udělený všemi uživateli po měsících

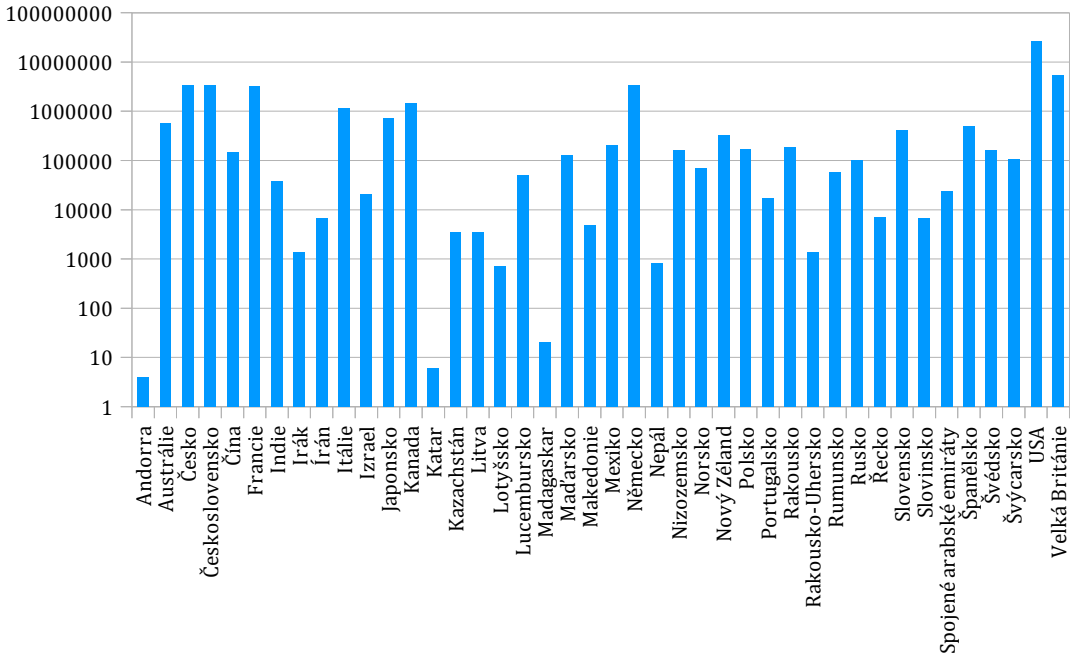


Obr. 3: Počet hodnocení udělený aktivními uživateli po měsících

Na následujících grafech je vidět počet hodnocení, které udělili uživatelé filmům podle žánru a vybraných zemí původu.



Obr. 4: Počet hodnocení dle žánrů filmů



Obr. 5: Počet hodnocení dle vybraných zemí původu filmu

Jak je vidět, zatímco počet hodnocení podle žánru filmu se liší maximálně o 3 řády, u rozdělení podle zemí původu může být rozdíl až 7 řádů.

3 Algoritmy

Naším cílem je predikce hodnocení, které by uživatel filmu udělil. Algoritmus je tedy funkce ***A (f, u) = r,*** která pro dvojici <film, uživatel> vrátí hodnocení ***r***. Výsledné hodnocení lze požadovat buď z celého rozsahu <0, 100>, nebo přímo z hodnotící množiny <0, 20, 40, 60, 80, 100>. Protože výsledná předpověď hodnocení bude prezentována uživateli, mohlo by se zdát, že lepší variantou bude přímo konkrétní množina, kterou můžeme znázornit ve formě hodnotících hvězdiček. Tato varianta by však zaokrouhlováním příliš znepřesnila výsledky a neumožnila by jemné ladění jednotlivých parametrů algoritmů. Proto budeme ***r*** požadovat z celočíselného rozsahu <0, 100> a teprve před prezentací uživateli bude provedeno zaokrouhlení a zobrazení.

Kvalita jednotlivých algoritmů bude posuzována jako střední kvadratická chyba (RMSE) podle následujícího vzorce:

$$x = \sqrt{\frac{1}{n}\sum_{n}^{i=1}(a_i - b_i)^2}$$ kde:

a je matice předpovězených hodnocení,

b je matice skutečných hodnocení pro stejnou množinu filmů a uživatelů,

n je počet testovaných hodnocení.

3.1 Algoritmus „Náhodného hodnocení“

Tento algoritmus předpovídá hodnocení zcela náhodně a slouží pouze pro srovnání s ostatními algoritmy. Aby bylo možné test provést opakovaně se stejným výsledkem, použijeme generátor pseudo-náhodných čísel. Využijeme pseudo-náhodný generátor, který je součástí programovacího jazyka Visual Basic [10]. Tento generátor je možné vždy před testem algoritmu nastavit do stejného počátečního stavu, čímž bude zaručeno, že bude dosaženo vždy stejných výsledků. Vlastní implementace algoritmu je následující:

```
alg_random (user_id, film_id) = random_get (0, 100)
```

Tento algoritmus má zcela zřejmě konstantní časovou i paměťovou složitost.

3.2 Algoritmus „Průměrného hodnocení“

Tento algoritmus vrací vždy průměrné hodnocení filmu, které bylo uděleno všemi ostatními uživateli (s podmínkou minimálně 500 ohodnocených filmů). Aby nebylo nutné vždy počítat průměrné hodnocení filmu, je možné si tuto hodnotu vypočítat předem a uložit ji do sumarizační tabulky s následující strukturou:

```
CREATE TABLE active_users_film_summary
(
  film_id integer NOT NULL,
  rating_count integer NOT NULL,
  rating_average numeric NOT NULL,
  CONSTRAINT active_users_film_summary_pkey PRIMARY KEY
(film_id)
)
WITH (
  OIDS=FALSE
);
```

Při každém přidaném nebo smazaném hodnocení od uživatele s více než 500 hodnoceními pak stačí upravit jeden řádek v této tabulce, což může zajistit trigger. Dále je potřeba vyřešit situaci, kdy uživatel ohodnotí pětistý film, případně smaže hodnocení a zbude mu pouze 499 hodnocení. Pak je potřeba upravit hodnocení všech 500 filmů, které hodnotil.

Při přidání nového hodnocení má trigger k dispozici veškerá potřebná data, takže není nutné znovu provádět dotaz nad tabulkou všech hodnocení. Stačí pouze upravit jeden řádek v sumarizační tabulce následujícím způsobem:

```
UPDATE active_users_film_summary
SET rating_count = rating_count + 1,
rating_average = (rating_average * rating_count +
new.rating) / (rating_count + 1)
WHERE film_id = new.film_id;
```

Obdobně lze postupovat při smazání hodnocení. Dotaz nad tabulkou hodnocení se bude provádět jedině v případě, že uživatel překročil limit 500 hodnocení.

S touto sumarizační tabulkou lze naimplementovat algoritmus průměrného hodnocení jednoduše jako dotaz na jeden řádek z této tabulky.

Zbývá vyřešit okrajovou podmínku, kdy k filmu ještě neexistuje žádné hodnocení. V tomto případě nelze využít existující hodnocení jako vodítko pro předpověď. Je možné takovému filmu hodnocení nepředpovědět, což je však v rozporu s původní potřebou konkrétního hodnocení. Filmu lze tedy předpovědět náhodné hodnocení, předem zvolené hodnocení, nebo je možné využít další známé parametry. Pokud by byly využívány jiné parametry, jako je žánr a původ filmu, získá tento algoritmus další závislosti, což není žádoucí. Nejlepším řešením tedy bude udělit filmu průměr hodnocení všech filmů na ČSFD.cz, což je 67,67%.

3.3 Algoritmus „Slope one“

Tento algoritmus byl představen v práci Daniela Lemire a Anny Maclachlan [9]. Algoritmus Slope One hledá podobnost mezi dvojicemi filmů podle rozdílu hodnocení uživatelů, kteří hodnotili oba tyto filmy.

Vztah dvou filmů $\boldsymbol{f}$ a $\boldsymbol{g}$ lze vypočítat podle následujícího vzorce:

$$R_{f,g} = \frac{\sum_{u \in U(f,g)} H_{u,g} - H_{u,f}}{|U(f,g)|}$$ kde:

U(f, g) je množina uživatelů, kteří hodnotili oba filmy ***f*** a ***g***,

$H_{u,f}$ je hodnocení filmu ***f*** uživatelem ***u***.

Nyní je možné předpovědět hodnocení filmu ***f*** uživatelem ***u***:

$$H_{u,f} = \frac{\sum_{g \in F(u)} ((H_{u,g} + R_{u,g}) * |F(u,g)|)}{|F_u|}$$ kde:

F(u) je množina filmů, které hodnotil uživatel ***u***, obdobně ***F(u, v)*** je množina filmů, které hodnotili oba uživatelé ***u*** a ***v***.

Výpočet lze demonstrovat na následujícím příkladu:

	Hodnocení filmu 1	Hodnocení filmu 2	Hodnocení filmu 3
Uživatel 1	20%	40%	-
Uživatel 2	80%	60%	100%
Uživatel 3	60%	-	60%

Pro zjištění vztahu mezi filmy 1 a 2, je nutné najít všechny uživatele, kteří hodnotili oba tyto filmy (tedy uživatel 1 a 2) a spočítat průměrný rozdíl hodnocení:

$$\frac{(40-20)+(60-80)}{2} = 0$$

Obdobně lze zapsat vztah mezi filmem 3 a filmem 2:

$$\frac{60-100}{1} = -40$$

Pro předpověď hodnocení filmu 2 uživatelem 3 je třeba spočítat vážené průměry hodnocení filmu 2 od všech ostatních filmů, které uživatel 3 hodnotil, tedy:

$$\frac{(60+0)*2+(60-40)*1}{2+1} = 46,\overline{6}$$

Algoritmus Slope one využívá předem vypočítaných vzájemných vztahů mezi filmy. Příprava těchto dat je sice výkonnostně náročná, ale protože jsou

počítána vzhledem ke všem uživatelům a jsou založena na velkém množství hodnocení, není nutné je přepočítávat příliš často. Při implementaci tohoto algoritmu by stačilo například každý den přepočítat vztahy mezi filmy pouze jednou. Předpovědi uživatelů by pak využívaly těchto dat.

Velkým problémem tohoto algoritmu je však množství vypočítaných dat, protože musí být uloženy vztahy mezi všemi filmy, což pro ***n*** filmů znamená $\frac{n*(n-1)}{2}$ uložených vztahů. Pro každý vztah je zapotřebí uložit minimálně tato data:

- ID zdrojového filmu (integer, 4 byte)
- ID cílového filmu (integer, 4 byte)
- rozdíl v hodnocení (real, 4 byte)
- počet hodnocení (integer, 4 byte)
- data pro vyhledávací strom (dle implementace cca 4 byte)

Následující tabulka ukazuje množství paměti, potřebné pro uložení vztahů mezi filmy. Je vidět, že množství potřebné paměti se zvyšuje s druhou mocninou počtu filmů.

Počet filmů	Množství uložených dat	Poznámka k počtu filmů
2	20 B	
130.000	157 GB	Počet ohodnocených filmů k 9. 1. 2012
300.000	838 GB	Počet všech filmů na ČSFD.cz
2.300.000	48 TB	Počet všech filmů na IMDB.com

Z uvedených dat je zřejmé, že algoritmus není škálovatelný pro velké množství filmů.

3.4 Algoritmus „k-nearest neighbor pro uživatele“

Toto je základní algoritmus používaný v oblasti doporučovacích systémů. Tento algoritmus se při předpovídání hodnocení filmu ***f*** uživatelem u snaží najít

k nejpodobnějších uživatelů k uživateli ***u***, kteří ohodnotili film ***f***. Výsledné doporučení je pak průměrem hodnocení ostatních uživatelů.

Podobnost dvou uživatelů (***Ψ***) lze spočítat různými způsoby. V tomto případě je nejvhodnější porovnat hodnocení filmů, které hodnotili oba uživatelé. Výsledná podobnost je pak definována následujícím vztahem:

$$\Psi = \frac{C + \sum_{f \in F} \left(R_{u_1, f} - R_{u_1, f} \right)}{|F|}$$ kde:

F je množina filmů, které hodnotili oba uživatelé,

$R_{u,f}$ je hodnocení, které uživatel ***u*** udělil filmu ***f***,

C je konstanta sloužící pro zvýhodnění podobnosti uživatelů s více společně ohodnocenými filmy; čím vyšší je tato konstanta, tím vyšší váha se přikládá množství společně ohodnocených filmů.

Výsledné doporučení je pak váženým průměrem hodnocení ***k*** nejpodobnějších uživatelů.

Zřejmou nevýhodou tohoto algoritmu je nutnost hledat vždy nejpodobnější uživatele, což je operace velmi náročná na výpočetní kapacitu. Je totiž potřeba porovnat hodnocení všech uživatelů s hodnoceními aktuálního uživatele. Složitost je tedy:

$O(u*r^2)$, kde:

u je počet uživatelů

r je průměrný počet filmů ohodnocených uživatelem

Pokud je tato data náročné vypočítat, nabízí se možnost mít tato data vypočítaná předem. To by znamenalo, že pro každého uživatele bude v databázi uloženo jemu nejpodobnějších ***s*** uživatelů. Poté by stačilo nalézt v těchto ***s*** uživatelích takových ***k*** nejpodobnějších uživatelů, kteří hodnotili hledaný film. Může se však stát, že v těchto ***s*** uživatelích hledaných ***k*** nejpodobnějších uživatelů vůbec nebude nebo v nich dokonce nebude žádný uživatel, který hledaný film hodnotil. Proto je potřeba zvolit ***s*** natolik vysoké (třeba i na úkor míry podobnosti), aby pro většinu filmů obsahovalo ***k*** nejpodobnějších

uživatelů, kteří daný film hodnotili. Nepodaří-li se přesto ***k*** nejpodobnějších uživatelů nalézt, bylo by nutné spokojit se s předpovědí z menšího množství uživatelů nebo pro tento případ znovu vyhledat nejpodobnější uživatele.

Další nevýhodou tohoto algoritmu je to, že po přidání jakéhokoliv hodnocení jsou zneplatněna veškerá data, protože uživatel, který hodnocení udělil, se mohl posunout v žebříčku podobnosti u všech ostatních uživatelů. V praxi samozřejmě tento posun nebude příliš veliký, ale rozhodně je třeba data přepočítávat velmi často.

kNN algritmy využívají při výpočtu ***n***-rozměrný prostor. V tomto případě je využíván ***f***-rozměrný prostor, kde ***f*** je celkový počet filmů. Jednotliví uživatelé zde představují body, mající na ose každého filmu takovou hodnotu, jaké udělili tomuto filmu hodnocení. Nevýhodou je, že algoritmy, používané pro indexování multidimenzionálních prostorů, mají problémy s velkým množstvím prostorů. V případě filmů na ČSFD.cz by se jednalo o prostor se 130.000 rozměry. Navíc je při počítání vzdálenosti v tomto prostoru nutné uvažovat speciální případ, kdy uživatel film nehodnotil, nemá tedy v tom rozměru žádnou polohu. Protože filmů v databázi je řádově více, než kolik je průměrný počet hodnocení uživatele, bude těchto případů naprostá většina.

3.5 Algoritmus „Film-DNA“

Tento mnou vyvinutý algoritmus se snaží potlačit nevýhody implementace standardního kNN algoritmu a zároveň přinést další výhody. Podstatou algoritmu je, že každému uživateli je spočtena tzv. *filmová DNA*, což je vektor, popisující uživatelův filmový vkus. Pomocí tohoto vektoru lze snáze hledat filmy, které by se uživateli mohly líbit a další uživatele s podobným vkusem. Navíc lze tento vektor využít k dalším účelům jako je zobrazení zjednodušeného uživatelova vkusu v jeho profilu, lepšího cílení reklamy a podobně.

Nejprve je však třeba definovat co to je filmový vkus. Ten označuje, jaký druh filmů se uživateli líbí a na jaké filmy se rád dívá. Neznamená to ale, že by všem filmům tohoto druhu uděloval vysoké hodnocení. Filmový vkus spíše určuje, že filmy s některými parametry se uživateli líbí více než ostatní.

Filmový vkus uživatele lze vypočítat tak, že si vytvoříme vektor, kde každá složka označuje jeden parametr a hodnotou je relativní odchylka průměrného hodnocení filmů s tímto parametrem od průměrného hodnocení všech filmů tímto uživatelem. Ta bude vyjadřovat, jaký má každý parametr vliv na hodnocení filmu uživatelem. Je však zcela zásadní, jaké parametry filmu zvolíme, protože některá rozdělení mohou mít pouze velmi malou rozlišovací schopnost. Dále je třeba počítat s tím, že dělení dle parametrů nemusí být disjunktní, tedy že jeden film může mít více parametrů. Pokud by byly jako parametry zvoleny například žánry filmu, pak jeden film může mít žánr „romantický“ i „komedie“ a zcela jistě existují uživatelé, kteří mají rádi „komedie“, ale nemají rádi „romantické“ filmy a naopak.

Filmový vkus neovlivňují pouze parametry filmu, ale i vlastnosti (parametry) uživatele, například pohlaví či věk. Při srovnání průměrného hodnocení filmů dle pohlaví uživatelů, lze u některých filmů vypozorovat značný rozdíl. Například seriál „Ordinace v růžové zahradě“ má průměrné hodnocení od mužů 10,14% a od žen 18,43%. Ve věkové skupině 10-20 let má průměrné hodnocení 17,56% a ve skupině 20-30 let 12,50%. Do vektoru filmového vkusu je tedy možné zařadit i parametry uživatele.

Výhoda Film-DNA algoritmu spočívá v tom, že v případě nedostatečné rozlišovací schopnosti některého z parametrů lze tento parametr snadno nahradit vhodnějším. Proti algoritmu kNN snižuje Film-DNA algoritmus radikálně počet dimenzí, ve kterých se vyhledávají nejoblíbenější uživatelé. Zatímco počet dimenzí algoritmu kNN je tvořen počtem filmů, u algoritmu Film--DNA je tvořen počtem položek (parametrů) ve vektoru filmové DNA. Počet dimenzí je použitím algoritmu Film-DNA snížen na konstantu, jejíž hodnotu lze navíc regulovat počtem zvolených parametrů filmové DNA.

Prvním krokem při výpočtu filmového vkusu je volba parametrů, které budou zahrnuty do vektoru filmové DNA. My v této práci zvolíme dělení dle žánrů a zemí původů filmů.

Pro vytvoření tabulky s vektory filmového vkusu *film_dna_relative* je vhodné nejprve vytvořit tabulku *film_dna_absolute* s průměrným hodnocením

filmu podle jednotlivých filmových parametrů. Pokud bude uložen s průměrným hodnocením i celkový počet hodnocení, ze kterého byl průměr vypočítán, bude snadné při přidání hodnocení upravit pouze konkrétní parametry. Procentuální odchylky v tabulce *film_dna_relative* jsou na základě hodnot v tabulce *film_dna_absolute* upravovány triggery, aniž by bylo třeba provádět dotazy nad tabulkou všech hodnocení. Tabulka *film_dna_absolute* může obsahovat pouze parametry, které jsou generované průměrným hodnocením. Parametry uživatele není třeba převádět na procentuální odchylku, proto mohou být uloženy až v tabulce *film_dna_relative*.

Vektor filmového vkusu může být uložen v databázi dvěma různými způsoby. Z databázového hlediska je preferovaný návrh se složeným primárním klíčem *user_id*, *parameter_id* jak ukazuje následující struktura:

```
CREATE TABLE film_dna_absolute
(
    user_id integer NOT NULL,
    parameter_id integer NOT NULL,
    value numeric NOT NULL,
    count integer NOT NULL,
    CONSTRAINT film_dna_absolute_pkey PRIMARY KEY (user_id,
parameter_id),
    CONSTRAINT film_dna_absolute_parameter_id_fkey FOREIGN KEY
(parameter_id)
    REFERENCES film_dna_parameters (id) MATCH SIMPLE
ON UPDATE RESTRICT ON DELETE CASCADE
)
WITH (
    OIDS = FALSE
);
```

Tato struktura umožňuje snadnou úpravu dat. Dále tato struktura umožňuje ukládat pro uživatele pouze ta data, která opravdu existují. Pokud tedy uživatel neviděl filmy s určitými parametry, nebude zapotřebí v databázi zbytečně zapisovat NULL hodnoty. Při použití takové struktury je však obtížnější vyhledávání podobných uživatelů. V tomto případě je třeba provést LEFT JOIN tabulky na sebe samu, jak ukazuje následující SQL dotaz:

```
SELECT user_id, SUM (abs (fd1.value – fd2.value)) AS distance
FROM film_dna_absolute fd1
LEFT JOIN film_dna_absolute fd2
ON fd1.parameter_id = fd2.parameter_id
WHERE fd1.user_id = in_user_id
AND ARRAY [fd1.parameter_id] && in_parameter_id
GROUP BY fd2.user_id
ORDER BY distance;
```

kde *in_user_id* je ID uživatele, pro něhož jsou hledáni podobní uživatelé a *in_parameter_id* je pole parametrů, podle nichž je hledáno. Toto je pouze zjednodušený dotaz, který neřeší prázdné hodnoty atd.

Je vidět, že databáze musí při provádění příkazu JOIN tabulku v paměti zkopírovat tolikrát, kolik parametrů je ve výpočtu použito. Mezivýsledek si navíc nemůže uložit do paměti, protože je pro každou kombinaci uživatele a parametrů jiný. Takový přístup se v praxi ukázal jako příliš pomalý pro produkční použití. Vyhledání nejpodobnějších uživatelů pro jednu předpověď trvalo v průměru 4s a velmi zatěžovalo databázi, což není na produkčním prostředí akceptovatelné.

Je zapotřebí navrhnout strukturu tabulky tak, aby databáze mohla datovou strukturu udržovat v operační paměti a byla využitelná napříč požadavky. Toho lze docílit tím, že každý uživatel bude mít v tabulce *film_dna_absolute* jediný záznam a každý parametr bude uložen ve zvláštním sloupečku. Struktura tedy bude vypadat následovně:

```
CREATE TABLE film_dna_absolute
(
  user_id integer NOT NULL,
  g1_average numeric,
  g1_count integer,
  g2_average numeric,
  g2_count integer,
  g3_average numeric,
  g3_count integer,
  …
  total_count integer,
  total_sum integer,
  CONSTRAINT film_dna_absolute_pkey PRIMARY KEY (user_id)
)
WITH (
  OIDS = FALSE
);
```

V takto definované struktuře lze navíc uložit i celkový počet započítaných hodnocení *total_count* a jejich součet *total_sum*, což zjednoduší výpočet procentuální odchylky v tabulce *film_dna_relative.*

Plnění takové struktury daty je náročnější, protože v textu SQL dotazů bude třeba používat přímo názvy parametrů. Toho lze dosáhnout pouze dynamicky generovanými SQL dotazy, vytvářenými například v uložených procedurách.

Celou tabulku lze vygenerovat s pomocí následujícího dynamického dotazu:

```
CREATE OR REPLACE FUNCTION create_table_film_dna_absolute ()
RETURNS void AS
$BODY$
DECLARE
  v_command text;
BEGIN
  EXECUTE '
  DROP TABLE IF EXISTS film_dna_absolute;
  SELECT ''CREATE TABLE film_dna_absolute (user_id integer
CONSTRAINT film_dna_absolute_pkey PRIMARY KEY NOT NULL, '' ||
array_to_string (array_agg (name), E'',\n'') || '', total_count integer,
total_sum integer)''
  FROM (
    (
      SELECT ''g'' || id || ''_average float, g'' || id || ''_count integer'' AS name
FROM genres ORDER BY id
    )
    UNION ALL (
      SELECT ''o'' || id || ''_average float, o'' || id || ''_count integer'' AS name
FROM origins ORDER BY id
    )
  ) _
  ' INTO v_command;
  EXECUTE v_command;
END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;

SELECT create_table_film_dna_absolute ();
DROP FUNCTION create_table_film_dna_absolute ();
```

Je zřejmé, že práce s takto strukturovanou tabulkou bude výrazně pracnější, ale na druhou stranu může taková struktura velice zrychlit vyhledávání podobných uživatelů. Nyní bude JOIN obsahovat na levé straně vždy pouze jeden řádek s hledaným uživatelem. Žádná pomocná datová struktura se v tomto případě nebude vytvářet, stačí seřadit tabulku podle vypočítané podobnosti uživatelů. Kromě toho si PostgreSQL automaticky ukládá ve vyrovnávací paměti data tabulek na disku, které byly používány v poslední době. Při prvním spuštěném dotazu bude tato tabulka načtena do operační paměti, a příští dotazy již nebudou potřebovat přistupovat na disk, budou tedy výrazně rychlejší.

Aby bylo možné toto zrychlení využít, je potřeba, aby se tabulka do operační paměti vešla. Pro každého uživatele jsou v tabulce uloženy pro každý parametr dvě numerické hodnoty, tedy přibližně 8 bytů. Pro 205 parametrů a 24.792 uživatelů zabírá tabulka přibližně 40MB. Při zvyšování počtu uživatelů roste paměťová náročnost lineárně, takže algoritmus Film-DNA lze snadno škálovat.

V rámci testování výkonnosti byly implementovány tabulky s oběma strukturami. Pokud měla databáze k dispozici dostatek operační paměti, aby mohla být celá tabulka v paměti, byl druhý přístup cca 350x rychlejší, než přístup první. Proto u všech dalších testů budeme používat pouze druhý přístup.

Tabulka *film_dna_relative* má velmi podobnou strukturu jako *film_dna_absolute*, pouze může obsahovat navíc parametry uživatele a nemusí obsahovat sloupečky *total_sum* a *total_count*, které slouží pouze pro výpočet procentuální odchylky.

Jednotlivé hodnoty parametrů se vypočítají podle následujícího vzorce:

$$PVR_{i,u} = PVA_{i,u} - \frac{total\_sum}{total\_count}$$ kde:

$\boldsymbol{PVR_{i,u}}$ je i-tý parametr relativního vektoru uživatele $\boldsymbol{u}$,

$\boldsymbol{PVR_{i,u}}$ je i-tý parametr relativního vektoru uživatele $\boldsymbol{u}$,

$\boldsymbol{total\_sum_u}$ je celkový součet hodnocení uživatele $\boldsymbol{u}$,

total_count*$_u$** je celkový počet hodnocení uživatele ***u.

Po vypočtení vektoru filmové DNA pro každého uživatele, lze pro zvoleného uživatele ***u*** snadno najít uživatele, kteří mají podobný názor na filmy se zvolenými parametry. Rozdíl vkusu dvou uživatelů pro konkrétní film je tvořen rozdílem jejich filmových DNA podle zvolených parametrů. Ten je možné spočítat podle následujícího vzorce:

$$\delta = \sum_{p \in P} |D_{u,p} - D_{v,p}| \quad \text{kde:}$$

P jsou zvolené parametry, podle kterých budeme porovnávat,

v je porovnávaný uživatel,

$\boldsymbol{D_{u,p}}$ označuje hodnotu parametru ***p*** filmové DNA uživatele ***u***.

Filmové DNA je možné porovnávat i jinými způsoby. Lze například započítat vliv počtu ohodnocených filmů v jednotlivých kategoriích. Toto porovnání vychází z předpokladu, že dva uživatelé, kteří sledují převážně filmy stejné kategorie, mají podobnější filmový vkus než uživatelé, kteří mají sice podobné průměrné hodnocení, ale každý z nich dává přednost filmům jiné kategorie. Tento vliv můžeme spočítat například následujícím způsobem:

$$\delta = \sum_{p \in P} |D_{u,p} - D_{v,p}| + |C_{u,p} - C_{v,p}| * i \quad \text{kde:}$$

$\boldsymbol{C_{u,p}}$ je procentuální podíl hodnocení, které uživatel ***u*** udělil filmům s parametrem ***p***,

i je konstanta, která určuje důležitost porovnávání počtů hodnocení vůči rozdílu hodnocení.

Při našem testu budeme mít možnost tento vliv započítat s různými hodnotami ***i***.

Pokud je již známa uživatelova DNA, zbývá předpovědět hodnocení filmu. To lze provést dvěma způsoby. Předpověď, založená na prostém porovnání parametrů filmu s uživatelovou DNA nebude příliš přesná. Bude spíše odpovídat uživatelově očekávání vůči nově uváděnému filmu s konkrétními parametry, o kterém zatím nezná žádné bližší podrobnosti. Tuto předpověď lze využít

pouze v případě, že k filmu zatím nejsou žádná další hodnocení. Pokud již jsou k dispozici hodnocení ostatních uživatelů, lze použít princip kolaborativního filtrování a nalézt uživatele, kteří mají nejpodobnější filmovou DNA a zároveň hodnotili hledaný film. Pro daného uživatele získáme předpověď vypočtením průměru z hodnocení tohoto filmu podobnými uživateli. My v našich testech vyzkoušíme jak aritmetický průměr, tak průměr vážený a porovnáme výsledky. Zároveň zkusíme měnit počet nejpodobnějších uživatelů, abychom našli optimální nastavení.

Důležitým parametrem algoritmu je bezesporu jeho složitost. I pokud by tento algoritmus dokázal předpovídat výborné výsledky, nebyl by v produkčním prostředí příliš užitečný, pokud by jeho výpočetní složitost byla příliš vysoká. Aby bylo možné posoudit složitost algoritmu Film-DNA, rozebereme jej po jednotlivých fázích:

1) Jednorázově je třeba spočítat filmové DNA pro všechny uživatele v systému. V této fázi se všechna hodnocení aktivních uživatelů zapíší do správné složky vektoru DNA konkrétního uživatele. Protože však jeden film může mít více parametrů, jedno hodnocení může být započteno vícekrát. Tato fáze má tedy složitost *O(r*v)*, kde ***r*** je celkový počet hodnocení a ***v*** je průměrný počet složek vektoru, do kterých náleží jeden film.

2) Při přidání nového hodnocení, jeho úpravě či smazání stávajícího hodnocení, je potřeba upravit DNA aktuálního uživatele. Protože v pomocné tabulce *film_dna_absolute* již je uložen celkový počet hodnocení, ze kterých se parametr počítal, lze jej upravit, aniž by bylo potřeba procházet další hodnocení. Tím, že se změní počet hodnocení, je však potřeba přepočítat celý řádek v tabulce *film_dna_relative*, což je však velmi rychlá operace se složitostí *O(w)*, kde ***w*** je počet složek vektoru.

3) Při změně parametru filmu je potřeba přepočítat filmovou DNA všech uživatelů, kteří tento film hodnotili. V algoritmu Film-DNA stačí přepočítat pouze tu složku, která se mění. Celková složitost je

$O(R_f * p * w)$, kde $\boldsymbol{R_f}$ je počet hodnocení filmu $\boldsymbol{f}$, $\boldsymbol{p}$ je počet ovlivněných parametrů a $\boldsymbol{w}$ je délka vektoru filmové DNA. Tato operace má lineární složitost závislou na počtu hodnocení filmu. U často hodnocených filmů je však výpočet náročnější. Parametry filmu se obvykle mění pouze výjimečně a zpravidla jen u nových (a tedy málo hodnocených) filmů.

4) Pro předpověď hodnocení filmu uživatelem, je třeba nalézt $\boldsymbol{n}$ uživatelů, jejichž vektor filmové DNA je nejpodobnější aktuálnímu uživateli. Základní algoritmus, který pouze porovná uživatelovu filmovou DNA se všemi ostatními uživateli má složitost $O(u*v)$, kde $\boldsymbol{u}$ je počet všech uživatelů v systému a $\boldsymbol{v}$ je počet složek vektoru, které chceme porovnávat. Pokud bychom použili některý z algoritmů na indexaci dat v $\boldsymbol{n}$-rozměrném prostoru, lze v ideálním případě zlepšit složitost až na $O(log(u))$. Indexace dat je však v takto rozměrném prostoru velmi obtížná. Možné problémy jsou blíže rozepsány v následující kapitole. V této práci proto využijeme prosté lineární porovnání všech uživatelů.

5) Ve chvíli, kdy najdeme $\boldsymbol{n}$ uživatelů s nejpodobnějším filmovým DNA, můžeme přistoupit k vlastní předpovědi hodnocení. Výsledná předpověď hodnocení bude tvořena průměrem hodnocení nalezených uživatelů na základě vzdálenosti jejich filmové DNA. Výpočet tohoto hodnocení má složitost $O(n)$, přičemž $\boldsymbol{n}$ je pevně zvoleno, časová složitost je proto konstantní.

Z uvedeného je zřejmé, že celková náročnost algoritmu závisí především na schopnosti najít $\boldsymbol{n}$ uživatelů, kteří mají filmovou DNA nejpodobnější filmové DNA aktuálního uživatele.

3.5.1 Problémy s indexací DNA dat

V případě, že porovnávání všech vektorů DNA při hledání nejpodobnějšího uživatele není dostatečně efektivní, lze využít některý z algoritmů na indexaci $\boldsymbol{n}$-rozměrných prostorů. Jejich základní varianty popsal ve své práci Peter N.

Yianilos [12], pro mnohé z nich již existují různé optimalizace. U všech těchto algoritmů se však jejich efektivita snižuje se zvyšujícím se počtem dimenzí. V aktuálním případě by bylo potřeba pracovat se 170 parametry, tedy vyhledávat ve 170 dimenzionálním prostoru. Pokud by bylo potřeba pracovat i s dalšími metadaty, jako je počet hodnocení s daným parametrem, tak se počet dimenzí ještě znásobí. V prostoru s takovým počtem dimenzí indexovací algoritmy pozbývají svoji efektivitu a reálná rychlost je srovnatelná s lineárním prohledáváním.

Tato indexace navíc velmi omezuje způsob, kterým lze vyhledávat. Bylo by třeba vyřešit například následující problémy:

1. Každá kombinace parametrů, podle které je potřeba vyhledávat, musí být opatřena samostatným indexem. Vyhledávání podobných uživatelů podle parametrů předpovídaného filmu by pak znamenalo mít $\binom{v}{p}$ indexů, kde $\boldsymbol{v}$ je počet složek vektoru DNA a $\boldsymbol{p}$ je maximální počet parametrů jednoho filmu. V našem případě by to znamenalo mít cca 15 bilionů indexů, což je zjevně nesplnitelné. Jediným řešením je udržovat pouze několik vybraných indexů a hledat vždy v tom indexu, který byl postaven podle parametrů nejpodobnějších aktuálnímu filmu. Tím je ale porušen koncept algoritmu, který dělí filmy podle parametrů na jednotlivé skupiny a sloučení různých kombinací ruší význam jednotlivých parametrů.
2. Při indexování dat standardními algoritmy, by podobnost uživatelů (tedy vzdálenost uživatelů v prostoru) musela být vždy počítána tak, aby splňovala pravidla Eukleidovského prostoru. To však brání použití dalších metadat při porovnání vzdáleností jednotlivých parametrů. Nelze například při porovnávání jednotlivých složek vektoru započítat množství ohodnocených filmů uživateli. Některá metadata lze rozvinout do další dimenze, avšak tento postup nelze aplikovat ve všech případech.

3. Jedním z problémů je práce s prázdnými daty. Je zapotřebí ošetřit případ uživatelů, kteří neviděli žádný film, hodnocený některým z použitých parametrů. Zde je nutné zvolit způsob výpočtu vzdálenosti těchto uživatelů a jejich indexace. Pokud by místo neznámých parametrů byla použita průměrná hodnota z hodnocení ostatních uživatelů, bude indexace jednoduchá, ale data budou zkreslena. Při použití průměrné hodnoty z hodnocení ostatních uživatelů, ale s poloviční váhou pro neznámé parametry by došlo k porušení Eukleidovského prostoru.

Je zřejmé, že indexování s sebou nese mnoho problémů. Naopak užitek, který by přineslo, není příliš výrazný. V některých speciálních případech by přínosy mohly převážit nevýhody, ale v našem konkrétním případě bude stačit prosté lineární prohledávání.

4 Výsledky

Testy byly prováděny na počítači s procesorem Intel Core2 Quad @ 2.40Ghz se 4 GB RAM na Microsoft Windows 7/64bit. Byla použita databáze PostgreSQL 9.2 s výchozím nastavením vyjma těchto údajů:

- shared_buffers = 2048MB
- work_mem = 1024MB

Každý algoritmus byl testován na množinách dat s různou velikostí a každý test byl proveden 3x. Výsledná délka běhu algoritmu byla vypočítána jako průměr posledních 2 měření. Tím se projevil i vliv cache podobně, jako při provozu na produkčním systému.

Všechny algoritmy byly implementovány jako databázová funkce *alg_<name>(user_id integer, film_id integer, ...)* se vstupními parametry ID uživatele, ID filmu, pro která se bude předpovídat hodnocení, a případně parametry ovlivňující vnitřní nastavení algoritmu.

Všechny SQL scripty pro přípravu dat a testování algoritmů jsou uloženy na přiloženém DVD.

4.1 Příprava dat

Pro testovávaní algoritmů bylo nutné připravit několik pomocných tabulek, aby byly výpočty rychlejší. Jak již bylo zmíněno dříve, rozhodli jsme se algoritmy testovat na 99% nejstarších hodnoceních a pouze na uživatelích, kteří mají v tomto období více než 500 hodnocení. Zbylé 1% hodnocení bude sloužit jako kontrolní vzorek pro ověření kvality předpovědí. Proto si vytvoříme tabulku *active_users* s ID všech uživatelů, kteří do testu budou zahrnuti.

Nejprve vytvoříme tabulku *active_users_film_ratings*, obsahující pouze hodnocení od uživatelů, vybraných pro testování. Tak se bude prohledávat pouze nezbytné množství dat. Tato tabulka obsahuje sloupeček *validation* typu

boolean, který může nabývat hodnoty *true*, pokud je záznam součástí kontrolního vzorku a *false* pokud je určen pro učení algoritmů.

Tabulka *active_users_film_summary* bude využívána dvěma způsoby:

- bude čtena sekvenčně podle ID pro předávání dat algoritmům a porovnání úspěšnosti předpovědí
- budou vyhledávána hodnocení uživatelů podle filmu, aby z nich algoritmy mohly počítat průměrné hodnocení

Nad tabulkou samozřejmě budou vytvořeny indexy, ale index při hledání hodnoty pouze vrátí ukazatel na stránku na disku, ve které je uložen řádek tabulky. Pokud je každá hledaná hodnota uložena v jiné stránce, je dotaz samozřejmě výrazně pomalejší, protože se musí načítat podstatně více dat z pevného disku. Pokud jsou všechna data načtena do cache v operační paměti, rozdíl se snižuje, ale stále je patrný. Proto je v našem případě vhodné vytvořit dvě tabulky: jednu seřazenou podle ID a druhou podle film ID. Tím se zajistí, že všechna hodnocení jednoho konkrétního filmu budou vždy na několika málo stránkách, takže přístup bude výrazně rychlejší.

4.2 Algoritmus „Náhodného hodnocení“

Pro tento algoritmus si vytvoříme sekvenci *random_seq*, která nám umožní generovat opakovatelná náhodná čísla. Pomocí této sekvence vytvoříme funkci *random_get (min, max)*, která vrátí celé náhodné celé číslo v intervalu <min, max> a funkci *random_reset ()*, která nastaví náhodný generátor do výchozího stavu.

Pro předpověď hodnocení využijeme tuto funkci:

```
CREATE OR REPLACE FUNCTION alg_random (in_user_id integer,
in_film_id integer)
RETURNS integer AS
$BODY$
    SELECT random_get (0, 100);
$BODY$
LANGUAGE sql VOLATILE
COST 10;
```

Vlastní test provedeme následujícím dotazem:

```
SELECT random_reset ();

SELECT sqrt (SUM ((alg_random (user_id, film_id) – rating) ^ 2) / COUNT
(*)) AS rmse, COUNT (*) AS count
FROM active_users_film_ratings
WHERE id IN (
    SELECT id FROM active_users_film_ratings WHERE validation = true
ORDER BY id LIMIT %count
);
```

kde %*count* je počet hodnocení k otestování.

Funkce nezávisí na žádných datech (vyjma sekvence *random_seq*), její časová složitost je konstantní a je proto velmi rychlá.

Výsledky jsou uvedeny v následující tabulce:

Velikost množiny	**RMSE**	**Doba běhu**
100	40,15	140ms
1000	40,03	150ms
10000	41,10	280ms

4.3 Algoritmus „Průměrného hodnocení“

Pro tento algoritmus si vytvoříme sumarizační tabulku *active_users_film_summary*, která obsahuje průměrná hodnocení všech filmů.

Algoritmus pak lze implementovat následovně:

```
CREATE OR REPLACE FUNCTION alg_average (in_user_id
integer, in_film_id integer)
   RETURNS integer AS
$BODY$
   SELECT round (COALESCE (rating_average, 67.67))::
integer
   FROM active_users_film_summary
   WHERE film_id = $2;
$BODY$
LANGUAGE sql STABLE
COST 100;
```

Je vidět, že funkce pro každou předpověď udělá jeden dotaz do tabulky podle primárního klíče. Funkce má opět konstantní časovou složitost a je rovněž velmi rychlá:

Velikost množiny	**RMSE**	**Doba běhu**
100	19,60	140ms
1000	18,21	160ms
10000	18,22	380ms

4.4 Algoritmus „User-DNA“

Pro použití tohoto algoritmu je třeba v databázi vytvořit tabulky *film_dna_absolute* a *film_dna_relative* podle popisu v kapitole Film-DNA. Pak lze vytvořit funkci, která bude předpovídat hodnocení:

```
CREATE OR REPLACE FUNCTION alg_film_dna (in_user_id
integer, in_film_id integer, in_rating_count integer,
in_count_multiple double precision, in_weighted boolean)
   RETURNS integer AS
$BODY$
DECLARE
   v_order_condition text;
   v_result integer;
BEGIN
   SELECT array_to_string (array_agg (query), ' + ') AS
query
   FROM (
    SELECT 'abs (d1.' || param_name || param || '_average
– d2.' || param_name || param || '_average) + ' ||
in_count_multiple || ' * abs (d1.' || param_name || param
|| '_count – d2.' || param_name || param || '_count)' AS
query
    FROM (
    (
         SELECT 'g' AS param_name, unnest (genre) AS
param
         FROM films
         WHERE id = in_film_id
    )
    UNION ALL
    (
         SELECT 'o' AS param_name, unnest (origin) AS
param
         FROM films
         WHERE id = in_film_id
    )
```

```
    ) _
   ) _
   INTO v_order_condition;

   IF v_order_condition IS NULL THEN
    -- there is no parameter in film that we can serach
by
    RETURN 68; -- return average rating
   END IF;

   EXECUTE '
    SELECT COALESCE (SUM (r.rating / (1 + u.distance)) /
SUM (1 / (1 + u.distance)), 67.67):: integer
    FROM film_ratings r
    JOIN (
         SELECT d2.user_id, ' || v_order_condition || '
AS distance
         FROM film_dna_relative d1
         JOIN film_dna_relative d2
         ON d1.user_id!= d2.user_id
         WHERE d1.user_id = $1
         AND d2.user_id IN (
              SELECT user_id
              FROM active_users_film_ratings_by_film_id
              WHERE film_id = $2
              AND validation = false
         )
         ORDER BY distance
         LIMIT $3
    ) u
    ON r.user_id = u.user_id
```

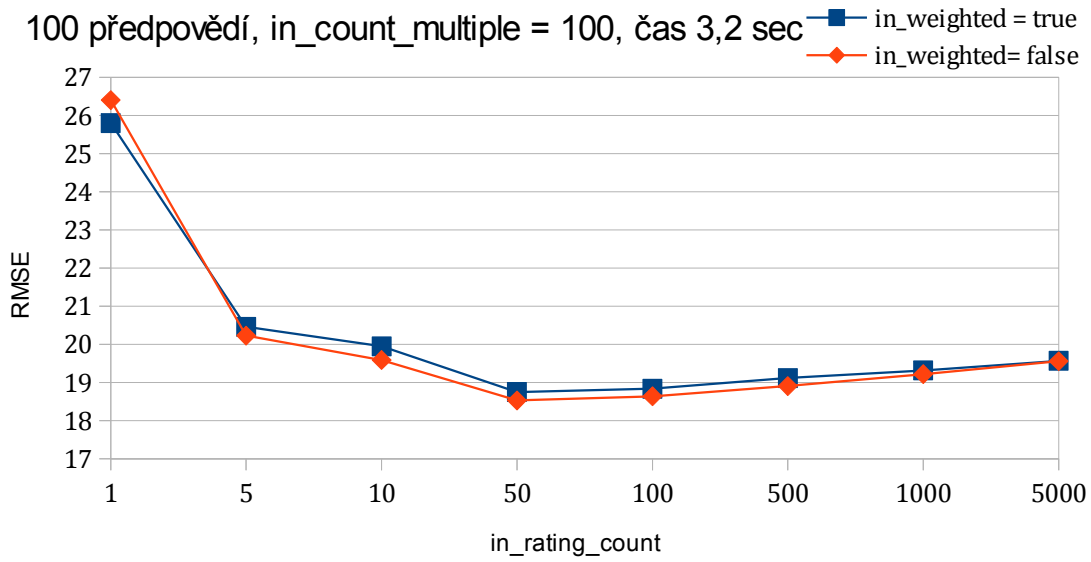
```
    WHERE r.film_id = $2' USING in_user_id, in_film_id,
in_rating_count INTO v_result;

   RETURN v_result;
END;
$BODY$
LANGUAGE plpgsql STABLE
COST 1000;
```

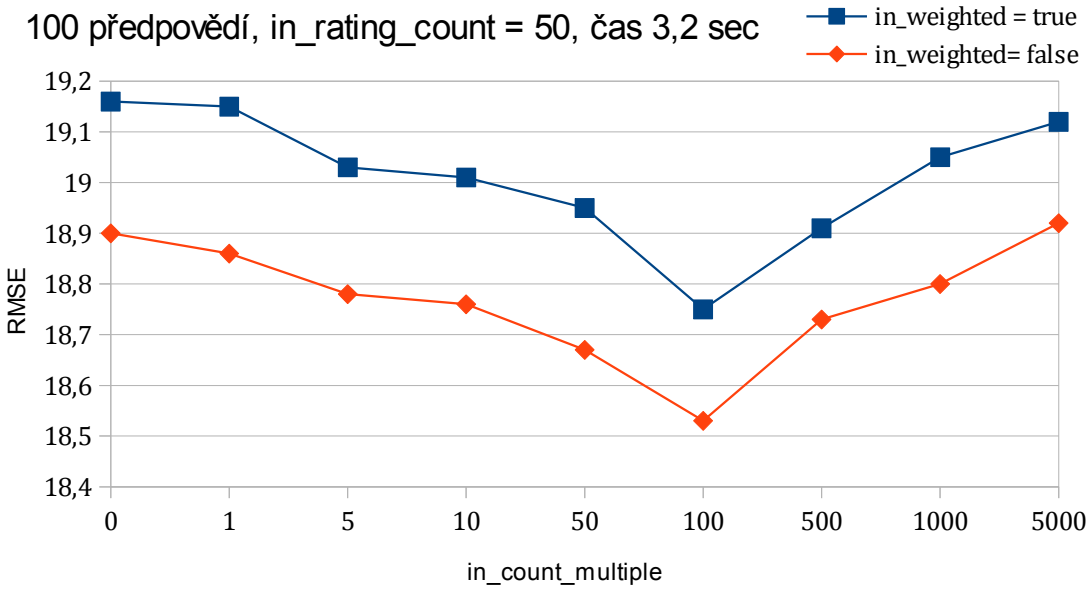
Tato funkce má jako vstupní parametry ID uživatele a ID filmu a také parametry, které nastavují její funkčnost:

- ***in_rating_count*** – Počet nejpodobnějších uživatelů, použitých pro předpověď.
- ***in_count_multiple*** – Multiplikátor, určující důležitost shody počtu hodnocení.
- ***in_weighted*** – Přepínač, určující použitou metodu průměrování. TRUE znamená vážený průměr, FALSE znamená aritmetický průměr.

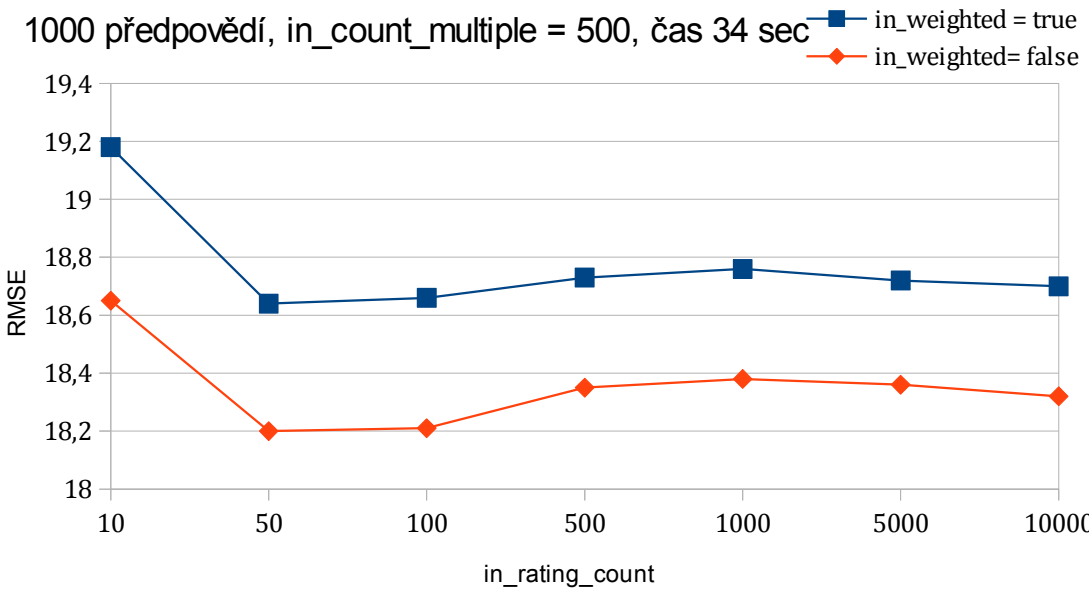
Cílem je nalézt takovou kombinaci parametrů, pro které bude RMSE co nejnižší. Pro dosažení optimální hodnoty RMSE je nutné otestovat různé kombinace těchto parametrů. Výsledky měření jsou v následujících grafech:



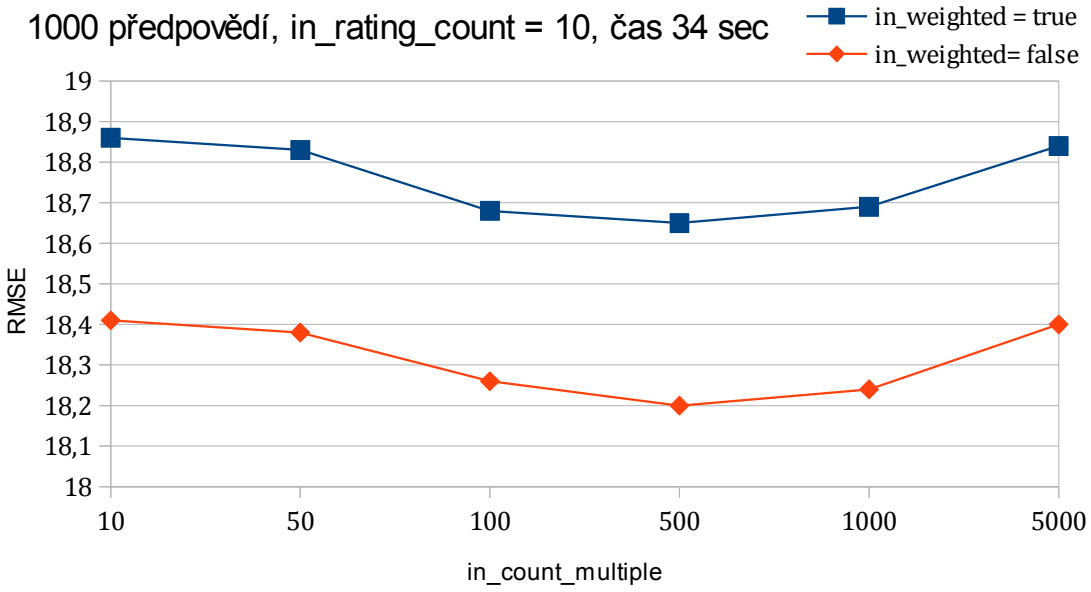
Obr. 6: Test vlivu proměnných ***in_rating_count*** a ***in_weighted*** při hodnotě ***in_count_multiple*** = 100 a 100 předpovědích.



Obr. 7: Test vlivu proměnných ***in_count_multiple*** a ***in_weighted*** při hodnotě ***in_rating_count*** = 50 a 100 předpovědích.



Obr. 8: Test vlivu proměnných ***in_rating_count*** a ***in_weighted*** při hodnotě ***in_count_multiple*** = 500 a 1000 předpovědích.



Obr. 9: Test vlivu proměnných ***in_count_multiple*** a ***in_weighted*** při hodnotě ***in_rating_count*** = 10 a 1000 předpovědích.

Z grafů je vidět, že nejlepší kombinací parametrů bylo:

- *in_rating_count* = **50**
- *in_count_multiple* = **100** – **500**
- *in_weighted* = **false**

4.5 Celkové výsledky

Pro názorné porovnání jednotlivých algoritmů jsou výsledky vypsány v tabulce:

Algoritmus	**Velikost množiny**	**RMSE**	**Čas**
Náhodné hodnocení	100	40,15	140 ms
Průměrné hodnocení	100	19,60	140ms
Film-DNA (50, 100, false)	**100**	**18,53**	**3,2s**
Film-DNA (50, 500, false)	**100**	**18,73**	**3,2s**
Náhodné hodnocení	1.000	40,03	150ms
Průměrné hodnocení	1.000	18,21	160ms
Film-DNA (50, 100, false)	**1.000**	**18,26**	**34s**
Film-DNA (50, 500, false)	**1.000**	**18,20**	**34s**
Náhodné hodnocení	10.000	41,10	280ms
Průměrné hodnocení	10.000	18,22	380ms
Film-DNA (50, 100, false)	**10.000**	**18,20**	**6,5min**
Film-DNA (50, 500, false)	**10.000**	**18,16**	**6,5min**

Je vidět, že se algoritmus „Film-DNA“ ve všech testovaných skupinách umístil na prvním místě, ale výsledek zdaleka není tak přesvědčivý, jak jsme mohli předpokládat. Dosažené hodnoty RMSE jsou téměř shodné s algoritmem „Průměrné hodnocení“, což je jeden z nejjednodušších algoritmů. Výpočetní náročnost algoritmu Film-DNA je přitom cca 1.000x vyšší než náročnost algoritmu „Průměrné hodnocení“.

I přesto tato práce přinesla užitečné výsledky. Oblast doporučování filmů je obecně velmi složitá, jak již například ukázala soutěž Netflix prize. Proto je každé zlepšení, nebo nový přístup k oblasti užitečný. Výpočetní náročnost

algoritmu Film-DNA je sice řádově vyšší, než u triviálních algoritmů, ale v produkčním prostředí je přesto dobře použitelná. Generování jedné předpovědi trvá 40ms, což je zcela dostačující. Hlavní výhodou je však modifikovatelnost. Triviální algoritmy jako „Průměrné hodnocení“ jsou velmi těžko rozšiřitelné. Pokud nestačí jejich přesnost, tak obvykle není možnost, jak ji zvýšit. Naopak algoritmus Film-DNA je velmi obecný a konfigurací zvolených parametrů můžeme zcela změnit předpovídání.

Algoritmus Film-DNA navíc nemusí být použit pouze pro předpovídání hodnocení filmů. Data, která jsou algoritmem vypočítána, nám říkají hodně o uživatelích. Pro každého uživatele tak máme informace o tom, jaké filmy často sleduje a jaké filmy se mu líbí, což můžeme využít například k přesnějšímu cílení reklamy. Dále je možné tato data využít pro funkci, která umožní uživatelům hledat jiné uživatele s podobným filmovým vkusem. V neposlední řadě je možné přihlášeným uživatelům zobrazovat v profilu ostatních uživatelů míru podobnosti jejich filmového vkusu.

5 Možná zlepšení

Přestože výsledky nedopadly podle očekávání, přinesla tato práce obecný algoritmus, který je snadno aplikovatelný pro konkrétní použití. Zlepšení výsledků by pravděpodobně bylo možné použitím jiných parametrů, jako jsou například významní tvůrci ve filmu. Lze předpokládat, že někteří uživatelé mají oblíbené herce a režiséry, jejichž filmům dávají přednost. Dále by bylo možné do vektoru uživatelova filmového DNA přidat údaje o konkrétním uživateli, jako je pohlaví, věk, atd. a při porovnávání uživatelů používat i tyto údaje.

Pro zásadní zlepšení výsledků algoritmu Film-DNA by bylo zapotřebí vyvinout způsob měření vhodnosti parametrů. Tímto způsobem by bylo možné eliminovat nevýhodné parametry a nahradit je novými. Postupný výběr nejvhodnějších parametrů by vedl ke zlepšení dosažené hodnoty RMSE. Jednou z možností, jak vhodnost parametrů testovat automaticky, jsou genetické algoritmy. Každý genetický algoritmus by představoval jednu konfiguraci parametrů vektoru. Konfigurace s dobrými výsledky by postoupily, mírně se upravily, zkřížily apod. Naopak konfigurace se špatnými výsledky už by dále nepostoupily a byly by zapomenuty. V tomto případě by však bylo potřeba dát si pozor na tzv. „přeučení“. Pokud by byly jednotlivé varianty testovány na stále stejném ověřovacím vzorku, naučily by se vybrat nejvhodnější nastavení parametrů pouze pro tento vzorek. Na jiných vzorcích by pak podávaly výrazně horší výsledky. Proto by bylo potřeba používat při každém testu jiný ověřovací vzorek.

V případě, že by se ukázalo, že každý parametr je jinak důležitý, lze do algoritmu doplnit váhy parametrů. Každý parametr by pak měl určenu svoji váhu na reálné škále <0, 1>. Vzorec pro výpočet podobnosti dvou uživatelů by pak vypadal takto:

$$\delta = \sum_{p \in P} |D_{u,p} - D_{v,p}| * W_p + |C_{u,p} - C_{v,p}| * i \quad \text{kde:}$$

$\boldsymbol{W_p}$ je váha parametru $\boldsymbol{p}$.

Další možností je upravit vstupní data pro tento algoritmus. Vzhledem k tomu, že filmový vkus se u většiny diváků v průběhu času mění, bylo by možné do učící množiny zahrnout např. pouze filmy ohodnocené v posledním půlroce, případně dávat nověji ohodnoceným filmům větší váhu.

Do funkce, porovnávající podobnost dvou uživatelů, lze přidat kromě váhy parametrů libovolná další metadata. Je například možné přidat do tabulky *film_dna_relative* informaci o rozptylu hodnocení uživatele. To nám může říci o uživateli další zajímavé informace, které můžeme využít při jeho porovnání s jiným uživatelem. Do vzorce na porovnávání dvou uživatelů je možné obecně přidat závislost na libovolných dalších metadatech.

Změnit by se mohl i výpočet relativních dat do vektoru. V současné době se počítají data relativně k vybranému uživateli. To například znamená, že vybraný uživatel preferuje jeden určitý žánr. Toto relativní porovnávání by bylo možné provádět ne vůči jednomu uživateli, ale napříč všemi uživateli. Význam této úpravy by spočíval v tom, že vybranému uživateli se žánr líbí více než ostatním uživatelům na ČSFD.cz. Tento způsob výpočtu by samozřejmě přinášel problém s aktualizací dat, protože každé ohodnocení filmu změní celé průměrné hodnocení ČSFD.cz a bylo by proto potřeba po každém ohodnocení přepočítat data všech uživatelů. V praxi to samozřejmě není nutné, protože v databázi ČSFD.cz je natolik velké množství dat, že by je stačilo přepočítávat například jednou ročně. Průběžné změny by byly totiž měly na kvalitu předpovědi zanedbatelný vliv.

6 Literatura

[1] Extracting Value from Chaos – John Grantz, David Reinsel, June 2011

[2] Amazon.com Recommendations: Item-to-Item Collaborative Filtering – Greg Linden, Brent Smith, Jeremy York, Jan/Feb 2003

[3] US Patent No.: US 6,266,649 B1

[4] Music Genome Project – http://www.pandora.com/about/mgp

[5] The BellKor Solution to the Netflix Grand Prize – Yehuda Koren, August 2009

[6] The BigChaos Solution to the Net, Andreas Töscher and Michael Jahrer, September 2009

[7] The Pragmatic Theory solution to the Netflix Grand Prize – Martin Piotte, Martin Chabbert, August 2009

[8] Netflix Recommendations: Beyond the 5 stars – Xavier Amatriain, Justin Basilico, April 2012 – http://techblog.netflix.com/2012/04/netflix--recommendations-beyond-5-stars.html

[9] Slope One Predictors for Online Rating-Based Collaborative Filtering – Daniel Lemire, Anna Maclachlan, April 2005

[10] INFO: How Visual Basic Generates Pseudo-Random Numbers for the RND Function - http://support.microsoft.com/kb/231847/en-us

[12] Data Structures and Algorithms for Nearest Neighbor Search in General Metrix Spaces – Peter N. Yianilos, 1993

[13] Large-scale Parallel Collaborative Filtering for the Netflix Prize - Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan, 2008

[14] Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions - Gediminas Adomavicius and Alexander Tuzhilin, June 2005

[15] Hybrid Recommender Systems: Survey and Experiments - Robin Burke, November 2002

[16] The Learning Behind Gmail Priority Inbox - Douglas Aberdeen, Ondrej Pacovsky, Andrew Slater, December 2010

[17] Large Scale Problem Solving with Neural Networks: The Netflix Prize Case -

Nicholas Ampazis, September 2010

[18] Content-based Recommender Systems: State of the Art and Trends - Pasquale Lops, Marco de Gemmis and Giovanni Semeraro, 2010

[19] MoviExplain: A Recommender System with Explanations - Panagiotis Symeonidis, Alexandros Nanopoulos, Yannis Manolopoulosr, 2009

[20] Extending Recommender Systems: A Multidimensional Approach - Gediminas Adomavicius, Alexander Tuzhilin, 2001

[21] A Field Guide to Genetic Programming - Riccardo Poli, William B. Langdon, Nicholas Freitag McPhee, March 2008